Model-based Management of Security Services in Complex Network Environments

João Porto de Albuquerque
Department of Informatics, University of Hamburg
Vogt-Koelln-Str. 30, 22527 Hamburg, Germany
Email: porto@informatik.uni-hamburg.de

Heiko Krumm
FB Informatik
University of Dortmund
44221 Dortmund, Germany

Paulo Lício de Geus
Institute of Computing
University of Campinas
13083-970 Campinas/SP, Brazil

***Abstract*—The security mechanisms employed in current networked environments are increasingly complex, and their configuration management has an important role for the protection of these environments. Especially in large scale networks, security administrators are faced with the challenge of designing, deploying, maintaining and monitoring a huge number of mechanisms, most of which have complicated and heterogeneous configuration syntaxes. Consequently, configuration errors are nowadays a frequent cause of security vulnerabilities. This paper summarizes results from a doctoral thesis that offers an approach to the configuration management of network security systems specially suited to the needs of the complex environments of today's organizations. The approach relies upon policy-based management and model-based management, extending these approaches with a modeling framework that allows the design of security systems to be performed in a modular fashion. The model is segmented into logical units (so-called *Abstract Subsystems*) that enclose a group of security mechanisms and other relevant system entities, offering a more abstract representation of them. In this manner, the administrator is able to design a security system—including its different mechanism types and their mutual relations—by means of an abstract and uniform modeling technique. A software tool supports the approach, offering a diagram editor for models. After the model is complete, the tool performs an automated policy refinement, deriving configuration parameters for each security mechanism in the system.**

I. Introduction

Current networked IT systems contain a series of heterogeneous security services, such as firewalls and secure content management systems, intrusion detection systems, identity management, key management and authentication systems, access control systems, log and audit systems, secure storage systems, virtual private network and cryptochannel systems. Services of different application types have different functionality and are usually supplied by different vendors. Therefore, the style, syntax and semantics of their configuration files differ from each other. Each service has to be configured according to particular settings; moreover, many services depend on each other, since they usually have to cooperate with each other in a consistent manner in order to enforce global security objectives. Among example scenarios are: the desired access of a given employee to a certain resource depending on suitable authentication (access control); packet filtering at a firewall and VPN connection settings. At least in large organizations, the settings are subject to frequent changes and tend to overstrain the IT system administrators. Indeed, a survey of three large-scale Internet sites concluded that 51% of all failures were caused by operator errors—configuration errors being the largest category of errors [1].

Thus, automation and homogenization are highly interesting challenges. Ideally, there should be one global abstract, high-level specification describing the objectives in an easy-to-understand, application-oriented and system independent way. Furthermore, there should be a tool to process the high-level specification and to automatically generate the corresponding configuration files. The approaches of policy-based management [2] and management policy hierarchies [3] yield a corresponding solution. In accordance with these approaches, the administrator task should be summarized as obtaining abstract high-level policies, which can then be refined into service- and system-dependent low-level policies. Finally, configuration files can be derived from those low-level policies.

Completely automated policy refinement, however, is not possible if high-level policies are to be kept as the sole input in the process, since system-specific details have to necessarily be considered during the refinement process. Therefore, our approach of model-based management (MBM) utilizes a hierarchically structured system model, which represents the networked IT systems on three interrelated levels of abstraction [4]. The high level policies are directly linked with the highest model layer. The automated policy refinement follows the system layer interrelations, relying upon the user-defined model in order to obtain the information necessary for generating the low-level policies. To that effect, MBM comes along with a modeling tool called Model-Based-Service-Configuration (MoBaSeC). It supports the interactive graphical modeling of object-oriented system models and policy representations in the form of object instance diagrams, in order to support the comfortable and easily maintainable definition of system models and high-level policy descriptions. A library of predefined model element class definitions—the so-called meta-model—and automated consistency checks facilitate the model construction. Aggregating folder objects support the handling of large sets of similar model elements like user identities and workstations. The modeling of the IT systems that comprise large organizations, nevertheless, results in broad models which, from the performance point of view, can be handled using MBM but, from the administrator's point of view, are—particularly on the lower layers—very complex and difficult to master.

However, the benefits of policy hierarchies and MBM still stand and are clearly the fundamental principles towards a scalable solution. To tackle the above problems we therefore came up with a divide-and-conquer approach to the modeling of network security systems through the so-called Abstract Subsystems (AS). The modeling of a given network segment is thereby subdivided into three steps: (1) the detailed definition of the network mechanisms—in the so-called Expanded Subsystem; (2) the establishment of an abstract view of those mechanisms by grouping them into abstract types, i.e. the Abstract Subsystems; and (3) the definition of the interaction among Abstract Subsystems in the Diagram of Abstract Subsystems (DAS). As such, the model designer must deal with the mechanisms' complexity only on a local basis for each subsystem, i.e. in steps (1) and (2), whereas in step (3) the system as a whole is considered by means of an understandable and abstract overall representation of the system's architecture. The MoBaSeC tool has been correspondingly extended to support the definition of DAS and ASs. Moreover, it is able to refine policies from the DAS layer into different ASs. The refinement relations that are applied have been formally verified to ensure that the system behavior resulting from the generated configuration upholds the abstract policies.

The DAS approach, its modeling principles and their applications are described in [5]. In [6] the policy support offered by the approach is analyzed and a formalism for the model is presented as a basis for achieving results about the validity of the policy refinement. These results are revised and further developed in the formal validation approach presented in [7]. On the other hand, [8] elaborates on the practical use of DAS in large-scale environments, presenting also the diagram visualization and manipulation techniques that were implemented in MoBaSeC to improve the handling of large models. The doctoral thesis summarized in the present paper [7] consolidates and extends the previous works.

The rest of this paper is organized as follows: Sect. II briefly discusses related work, while Sect. III describes the modeling technique of *Abstract Subsystems* and gives a simple model example. In Sect. IV the tool support offered is presented and the automated policy refinement process is discussed. Sect. V presents in turn experimental results from several case studies performed. Lastly, Sect. VI casts conclusions for this paper.

II. Related Work

Though many of the firewall commercial products include configuration tools, they do not seem to have integrated security management as a primary concern. Nevertheless, a series of market product tools in this direction is slowly appearing, for example, *Checkpoint*'s INSPECT visual policy editor [9], and the product-independent *Solsoft NP* Tool [10].

As for research work, there are some approaches which draw on logic-based languages to specify security policies (e.g. [11]). Although such approaches offer advantages in scalability gains and detailed analysis capabilities, the syntax used is far from both the administrator's view of the system and the business view of the problem, requiring extra training for the correct policy specification. The graphical tool Firmato [12] offers a more intuitive approach, since it supports an interactive policy design by means of diagrams and automatically derives the corresponding configurations for mechanisms. However, this and the other approaches above proceed bottom-up, starting from the mechanisms to be configured and achieving representation languages that are too bound up with the mechanism types initially considered.

On the other hand, there are already some approaches that turn the problem right side up by starting from convenient abstract models that are able to address the management problem from a business point of view. The Power prototype [13] is one such approach that aims at supporting the building of policy hierarchies by means of a tool-assisted policy refinement based on templates. The specification of templates is done in a prolog-like language, thus suffering from the same problems of the aforementioned logic-based approaches.

In a wider context, Damianou *et al.* [14] present a set of tools for the specification, deployment and management of policies specified in the PONDER language. A further work by these authors offers an approach to the implementation and validation of PONDER policies for Differentiated Services using the DMTF CIM. to model network elements [15]. CIM concentrates on the modeling of management information (e.g. device's capabilities and state) while our model represents the whole relevant structure of the managed system together with the management components, producing a graphical representation understandable to the security administrator.

III. Modeling Technique and Application Example

Our modeling builds upon the Model-based Management approach [4] and employs a three-layered model whose structure is shown in Fig. 1. The horizontal dashed lines of the figure delimit the abstraction levels of the model: *Roles & Objects* (RO), *Subjects & Resources* (SR), and *Diagram of Abstract Subsystems* (DAS). Each of these levels is a refinement of the superior one in the sense of a "policy hierarchy" [3]; i.e. as we go down from one layer to another, the abstract system's view contained in the upper level is complemented by the lower-level system representation, which is more detailed and closer to the real system. As for the vertical subdivision, it differentiates between the model of the actual managed system and the security policies that regulate this system.

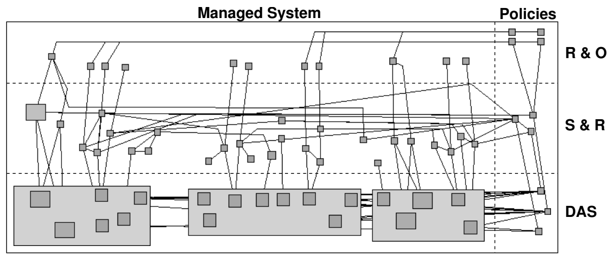

Fig. 1. Model Overview

As the lowest level of the model (DAS) is the focus of the present work, it is explained in detail in the next section. The two uppermost levels (RO and SR) have been adopted from previous work on model-based management, and thus will be presented briefly.

The RO level is based on concepts from Role-Based Access Control (RBAC) [16]. The main classes in this level are: *Roles* in which people who are working in the modeled environment act; *Objects* of the modeled environment that should be subject to access control; and *AccessModes*; i.e. the ways of accessing objects. The class *AccessPermission* expresses a security policy, allowing the performer of a *Role* to access a particular *Object* in the way defined by *AccessMode*.

The second level (SR in Fig. 1) offers a system view defined on the basis of the services that will be provided, and it thus consists of a more complex set of classes. Objects of these classes represent: (a) people working in the modeled environment (*User*); (b) subjects acting on the user's behalf (*SubjectTypes*); (c) services in the network that are used to access resources (*Services*); (d) the dependency of a service on other services (*ServiceDependency*); and lastly (e) *Resources* in the network. At this level, security policies are represented by *ServicePermissions* objects, each one of which expresses an authorisation for a *SubjectType* (on behalf of a *User*) to use a *Service* to access a *Resource*.

A. *Diagram of Abstract Subsystems*

The main objective of the Diagram of Abstract Subsystems (DAS) is to describe the *overall structure* of the system in a modular fashion; i.e. to cast the system into its building blocks and to indicate their interconnections. As such, a DAS is, formally speaking, a graph comprised of Abstract Subsystems (ASs) as nodes and edges that represent the possibility of bi-directional communication between two ASs. An AS, in turn, contains an abstract view of a certain system segment; i.e. a simplified representation of a given group of system components that may rely on the following types of elements:

Actors:	groups of individuals which have an *active* behavior in a system; i.e. they initiate communication and execute mandatory operations according to *obligation policies*.
Mediators:	elements that intermediate communication, in that they receive requests, inspect traffic, filter and/or transform the data flow according to the *authorization policies*; they can also perform mandatory operations based on *obligation policies*, such as registering information about data flows.
Targets:	*passive* elements; they contain relevant information, which is accessed by *actors*.
Connectors:	represent the interfaces of one AS with another; i.e. they allow information to flow from, and to, an AS.

Each element of the types *Actors*, *Mediators* or *Targets* represents a group of system elements that have a relevant behavior for a global, policy-oriented view of the system. As

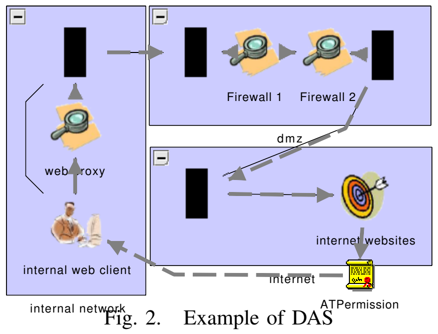

Fig. 2. Example of DAS

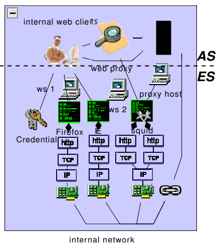

Fig. 3. Expanded AS

for the *Connectors*, they are related to the physical interfaces of an AS (for a detailed explanation on the modeling of abstract subsystems we refer to [5]). In this manner, a DAS supports the reasoning about the structure of the system *vis-à-vis* the executable security policies, thus making explicit the distribution of the different participants of these policies over the system.

Furthermore, in order to model the security policies themselves, another object type is also present in a DAS: *ATPermissions* (ATP). An ATP is associated with a path (p) in a DAS that connects an *Actor* (A) to a *Target* (T), possibly containing a number of *Mediators* and *Connectors* along the way. It expresses the permission for p to be used by A in order to access T. In this manner, each ATP models an *authorization policy*. The ATP objects in the system are not defined by the modeler, but rather derived automatically in a process that is explained below.

Additionally, each AS in a DAS is also associated with a detailed view of the system's actual mechanisms, called Expanded Subsystem (ES). This expanded view encompasses objects that represent hosts, processes, protocols and network interfaces of the system and supports the process of automated generation of configuration parameters (see Sect. IV).

B. *Example of DAS*

An example of DAS is shown in Fig. 2. This diagram corresponds to a simple network environment with three ASs: `internal network`, `dmz` (*demilitarized zone*) and `external network`. In the `internal network`, the object `internal web clients` is an *Actor* representing a group of processes that are authorized to access the processes mapped by the *Target* `internet web sites` (in `external network`) through the *Mediator* `Web proxy`, by the ATP `int_clients may surf on inet_sites`.

Figure 3 shows both the abstract and the expanded view for the AS `internal network` (leftmost of Fig. 2). Each object in the abstract view of the AS is then related to the elements that model the corresponding real entities of the system; for instance, the *Actor* `internal web clients` is associated with its respective *Process*-typed objects. This double representation of an AS provides the designer with a flexible model of the system that offers not only a more abstract, concise and understandable description of the system's structure, but also a detailed view of its mechanisms. It also constitutes the basis for applying the techniques presented in the next section.

IV. Tool Support and Automated Refinement

As explained in Sect. III, in the Model-Based Management approach, a software tool firstly assists the modeling by means of a diagram editor, and thereafter it automatically derives lower-level policies based on the model entities defined. This process is supported by a generic tool called MoBaSeC (Model-Based Service Configuration). The architecture of MoBaSeC is defined in such a way so that different applications of MBM are covered by the same common tool. The details of each application are defined by means of a meta-model, i.e. a set of classes that specify the node classes of each layer, the allowed connections between classes, and the consistency rules to which each model instance must comply. As such, the same basic tool (MoBaSeC) can be used to support the management of an arbitrary application context, as long as there is a meta-model that defines the structure for models in that particular context.

A. Model Editing, Visualization and Navigation

MoBaSeC is implemented in Java and basically consists of an object-oriented graph editor. To define a model instance, the user simply selects one of the classes available in the meta-model and creates a new object of this class in a window that contains a view of the model instance. Connections between two objects are thus established by dragging-and-dropping edges to connect the objects. During the modeling, the tool checks the consistency rules defined in the meta-model in order to ensure that the model instance is valid.

Furthermore, the tool prototype also implements *focus & context* techniques in order to substantially improve the navigation and visualization of large models (for further details see [8]). Through a combined use of the techniques *fisheye view* and *semantic zooming* associated to the modeling framework developed here, a larger focused area is made possible even if the context of the model is still visible, which leads to an optimization of the screen space. This gives the designer the capability of defining in detail a certain part of the model without losing sight of the system as a whole.

B. Configuration Process and Automated Policy Refinement

Having as input the abstract policy statements and the intended network scenario, the configuration development process evolves through the following steps:

1) modeling of the abstract policy (RO layer of Fig. 1),
2) definition of the system services and their relations to the abstract elements of the preceding step (SR layer of Fig. 1),
3) modeling of the relevant system elements by means of their abstract representations (i.e. the Abstract Subsystems) as well as their relations to the objects of the service-oriented system's view,
4) modeling of the Expanded Subsystem for each Abstract Subsystem in accordance with the real network,
5) automated refinement of the high-level policies at the RO level through the other abstraction levels, culminating in the generation of the low-level configuration parameters for the security mechanisms.

Therefore, after a valid model instance is input, the supporting tool automatically builds a policy hierarchy by deriving lower-level policy sets from the policies specified at the most abstract layer. This process is termed in the literature *policy refinement* [3].

In our approach, in the uppermost model level (RO) authorization policies are represented by *AccessPermissions* (Sect. III-A). The set of *AccessPermissions* is given by the modeler and acquires in this context a particular meaning. On one hand it defines the explicit permission for *Roles* to access *Objects* (in the way defined by an *AccessMode*)—corresponding to positive authorization policies. On the other hand, we adopt closed policies, i.e. the default decision of the reference monitor is denial. This implies that all triples of *Role*, *Object* and *AccessMode* not belonging to the set of *AccessPermissions* are forbidden. They thus implicitly define the negative authorization policies which the security mechanisms must as well enforce.

The automated refinement of authorization policies thus starts from the analysis of *AccessPermissions* in the RO level and their related objects, in order to generate a set of corresponding permissions in the SR level. Thus, each triple of *Role*, *AccessMode* and *Object* (r, am, o) related to an *AccessPermission* produces a set of 4-tuples (u, st, sv, r), each of which expresses an authorization for a *SubjectType* on behalf of a *User* to use a *Service* in order to access a *Resource*.

Subsequently, the *ServicePermissions* are refined into *ATPermission* objects (actor-target permissions, ATP for short, see Sect. III). Since ATPs are paths in the DAS graph, this refinement phase consists of, for each *ServicePermission* (u, st, sv, r), finding the shortest path between each *Actor* that is connected to the pair (u, st), and each compatible *Target* that is connected to a pair like (sv_t, res).

The following refinement phase comprises the automated generation of policies that takes into account the equipment and security mechanisms defined in the ESs. For this purpose, the tool generates, for each ATP, a corresponding *Allowed Expanded Path* (AEP) that represents an authorized path in the expanded subsystem views. Each AEP connects a process (that refines an *Actor*) to other processes (that refine the *Mediators* and the *Target*) through their related protocol stack, interface, and network objects.

In the last phase of the refinement process, a special back-end function for each mechanism type supported is executed. It analyzes the characteristics of each AEP that passes through the mechanisms of a type (such as communication protocols, addresses and ports) to produce corresponding low-level, device-dependent configuration parameters. Clearly, the configuration for a given mechanism must allow only the accesses corresponding to the AEPs that traverses the mechanism.

C. Refinement Correctness

In an automated policy refinement, only if the policy sets at the different levels are in perfect harmony, one can trust the system behavior resulting from the application of the lowest-level policies to be in conformance with the specified abstract goals. In pursuing this goal, we have developed a formal approach to the validation of the policy hierarchies generated in this work. Following the literature on multi-layered policy hierarchies and considering the special meanings conveyed by policies in our approach, two correctness criteria were selected: a) *Completeness*, i.e. the desired behavior specified in an abstract manner is completely implemented at the lower levels; b) *Consistency*, i.e. none of the actions enabled at the lowest level contradict the desired behavior of the highest level specification. As such, *completeness* warrants compliance with the explicit positive policies, whilst *consistency* makes sure that the implicit negative policies are enforced.

Based on these criteria, a number of condition sets were established that concern: i) the relation between elements of policy sets of two adjacent abstraction levels (*refinement conditions*); and ii) system objects at a given abstraction level as compared both with system objects at the immediate superior level, and with policies at the same layer (*structural consistency conditions*). Such conditions were defined for each of the refinement phases described above.

Subsequently, in order to be able to reason about the system behavior that results from the application of the generated configuration, *model representativeness axioms* were defined that formalize the implicit assumptions behind the model. These axioms state assumptions concerning both the mapping of real world entities by the model objects (*correct modeling*) and the effective application of normative aspects in the model to the real system (*correct implementation*). Relying upon axioms and conditions, we are able to prove two *validation theorems*:

- *VT1*: *For each high-level policy, the system enables all accesses in the real world that correspond to the policy*;
- *VT2*: *To each possible access in the real world there is a corresponding high-level policy*.

The two theorems correspond to the elected validation criteria of *completeness* and *consistency*, so that the meaning of the formal proofs is: once a model complies with the conditions defined and the axioms hold, the refinement process is *correct*; i.e. the generated system configuration is in conformance with the abstract policies so defined.

The supporting tool checks a group of these validation conditions on-the-fly as model objects are defined and on user's demand whenever s/he chooses the menu option 'Check Model'. A second condition group is tested along with the refinement process, and yet another group is applied after the refinement. These conditions are thus capable of validating the multi-layered structural architecture of the system *vis-à-vis* the security policies prescribed.

V. Case Studies and Experimental Results

The approach has shown its practical relevance in a series of case studies. It has been employed to the integrated configuration management of packet filters and VPN gateways of realistic network environments with different number of network elements and growing security policy complexity. Back-end functions were implemented for the generation of configuration files of the corresponding mechanisms of the OpenBSD operating system (pf and isakmpd), successfully covering the basic functionalities of these mechanisms.

To give a flavor of the models achieved, Figure 4 presents the DAS model for a network scenario of an enterprise network, composed of a main office and branch office, that is connected to the Internet. The DAS is composed by five subsystems, representing the logical network segments of the described scenario: `internal network`, `dmz`, `Internet`, `branch office's network` and `remote access point` (for further details see [5]).

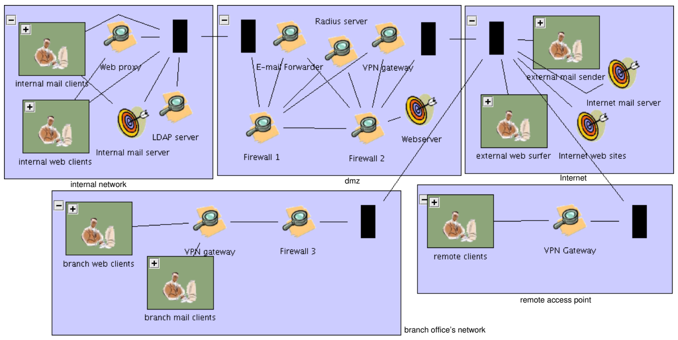

Fig. 4. DAS for a realistic network environment

The model development proposed in our approach can be used not only in a top-down fashion (starting from the modeling of the most abstract policies and proceeding downwards to the definition of the actual system elements) but also in a bottom-up one. Sometimes the existence of a network element might be the evidence of the need for an additional policy not yet modeled. Conversely, experience has shown that in practice the modeled network systems are frequently not capable of enforcing the given high-level policies. In such cases, the validation conditions described in Sect. IV-C cannot be satisfied, and the tool indicates the model elements which violate the policies. In this manner, by the combined use of top-down and bottom-up approaches, the modeler receives valuable information to do the necessary modifications on the model in order to achieve a system configuration that is congruous to the high-level policies.

The case studies performed clearly show the advantages brought forth by the DAS level based on information hiding and modularisation. Indeed, in Fig. 4 each Abstract Subsystem hides details about its inner features (i.e. network topology, hosts, processes, protocols etc.), providing instead a policy-oriented view of them that allows the administrator to reason about the distribution of policy participants over the system. As such, in Fig. 4 the modeler can reason about the system through a concise view of 32 DAS elements instead of the 540 actual network elements [5].

Furthermore, by comparing the different case studies performed we can reason about the growth behavior of the DAS level when moving from simpler scenarios to larger and more complex ones. This comparison has shown that the size of a DAS does not increase in the same pace as the number of elements in the ES level does (and thus as the system's mechanisms and number of required configuration rules do); rather, it does so at a much slower rate [5].

As can be derived from the figures above, our model provides a significant complexity reduction for the security administration task, since the number of entities to be dealt with are an order of magnitude or more smaller for these scenarios. For very large networks, say thousands of network nodes, the advantages can be even more impressive, as the number of different security scenarios tends to level out. This makes clear the scalability gain afforded by DAS in the support of large and complex models in comparison to previous work.

Since the development of the detailed IT system model, however, still demands considerable efforts, future work shall provide for automated modeling based on network exploration through management protocols, such as SNMP. Nonetheless, it is anticipated that the modeling effort will be reduced in applications of the technique to other environments, since the models already achieved can be used as templates.

VI. Conclusion

This work has proposed a design process for the configuration management of network security systems that is especially concerned with complex environments, aiming to bridge the gap between high-level security policies and the configuration parameters of employed mechanisms.

The main contributions of the thesis presented in this paper can be summarized as follows:

- a modeling technique for the management of security systems that achieves scalability by the segmentation of the system in *Abstract Subsystems*, enabling the process of model development and analysis to be performed in a modular fashion;
- tool support for comfortable model editing that implements *focus and context* techniques to improve the handling of large models, allowing the designer to define in detail a certain model part without losing sight of the system as a whole;
- a multi-layered modeling process in which the abstract policy representation is gradually brought into a more concrete system view, bridging the gap between high-level security policies and real system implementations;
- an automated policy refinement process that suits the needs of large-scale, complex environments and derives lower-level policies from the abstract policy set defined by the modeler, eventually yielding configuration parameters for security mechanisms;
- a validation approach for the policy refinement, which includes a formalism of the model, the establishment of consistency conditions, and the proof of the refinement correctness;
- representative case studies that show the practicality and the concrete advantages brought forth by this work.

As for future work, the scope of the high-level policies proposed here could be broadened to include other requirements, such as reliability and performance.

References

[1] D. Oppenheimer, A. Ganapathi, and D. Patterson, "Why do internet services fail, and what can be done about it," in *4th USENIX Symposium on Internet Technologies and Systems (USITS'03)*, 2003, pp. 1–16.

[2] M. Sloman, "Policy driven management for distributed systems," *Journal of Network and Systems Management*, vol. 2, no. 4, pp. 333–360, 1994.

[3] J. D. Moffett and M. S. Sloman, "Policy hierarchies for distributed system management," *IEEE JSAC Special Issue on Network Management*, vol. 11, no. 9, pp. 1404–1414, December 1993.

[4] I. Lück, S. Vögel, and H. Krumm, "Model-based configuration of VPNs," in *Proc. 8th IEEE/IFIP Network Operations and Management Symposium NOMS 2002*, R. Stadtler and M. Ulema, Eds. Florence, Italy: IEEE, 2002, pp. 589–602.

[5] J. Porto de Albuquerque, H. Krumm, and P. L. de Geus, "On scalability and modularisation in the modelling of security systems," in *Computer Security - ESORICS 2005, 10th European Symposium on Research in Computer Security*, ser. Lecture Notes in Computer Science, S. D. C. di Vimercati, P. F. Syverson, and D. Gollmann, Eds., vol. 3679. Berlin Heidelberg, Germany: Springer-Verlag, September 2005, pp. 287–304.

[6] ——, "Policy modeling and refinement for network security systems," in *6th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2005)*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 24–33.

[7] J. Porto de Albuquerque, "Gerenciamento baseado em modelos da configuração da segurança de ambientes de rede complexos [model-based management of the security configuration in complex network environments]," Ph.D. dissertation, Institute of Computing, University of Campinas. Available at: http://www.ic.unicamp.br/~jporto/research.html, Campinas, Brazil, September 2006.

[8] J. Porto de Albuquerque, H. Isenberg, H. Krumm, and P. L. de Geus, "Improving the configuration management of large network security systems," in *Ambient Networks: 16th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2005*, ser. Lecture Notes in Computer Science, J. Schönwälder and J. Serrat, Eds., vol. 3775. Berlin Heidelberg, Germany: Springer-Verlag, October 2005, pp. 36–47.

[9] *Check Point Software Revolutionizes Internet Security Management*, Check Point Software Technologies Ltd., Redwood City, Calif., January 2000.

[10] *Solsoft NP: Putting security policies into practice. White Paper*, Enterprise Management Associates, 2000, http://www.solsoft.com/library/ema_profiler.pdf.

[11] J. D. Guttman, "Filtering postures: local enforcement for global policies," in *SP '97: Proc. of the 1997 IEEE Symposium on Security and Privacy*. Washington, DC, USA: IEEE Computer Society, 1997, p. 120.

[12] Y. Bartal, A. J. Mayer, K. Nissim, and A. Wool, "Firmato: A novel firewall management toolkit," *ACM Transactions on Computer Systems*, vol. 22, no. 4, pp. 381–420, November 2004.

[13] M. Mont, A. Baldwin, and C. Goh, "POWER prototype: Towards integrated policy-based management," in *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS2000)*, J. Hong and R. Weihmayer, Eds., Hawaii, USA, 2000, pp. 789–802.

[14] N. Damianou, N. Dulay, E. Lupu, M. Sloman, and T. Tonouchi, "Tools for domain-based policy management of distributed systems," in *IEEE/IFIP Network Operations and Management Symposium (NOMS2002)*, Florence, Italy, 2002, pp. 213–218.

[15] L. Lymberopoulos, E. Lupu, and M. Sloman, "Ponder policy implementation and validation in a CIM and differentiated services framework," in *IFIP/IEEE Network Operations and Management Symposium (NOMS 2004)*, Seoul, Korea, April 2004.

[16] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, 1996.